



# Network Programming

by `SyntaxError`

# Table of Content

Chapter 1 : Internet Protocol Suite	3
🔊 Introduction to internet Protocol Suite	3
🔊 TCP/IP Layer Description	3
🔊 IP (Internet Protocol) Address	5
🔊 Routing Schemes	6
Chapter 2: Basic Socket Programming	7
🔊 Introduction to Basic Socket Programming	7
🔊 Connection-oriented vs Connectionless	7
🔊 Synchronous vs Asynchronous	7
🔊 System.Net.Sockets Namespace	8
🔊 How to make a connection ( Synchronous )	8
🔊 How to send and receive data	11
Chapter 3: Application Protocol	16
🔊 Introduction to Application Protocol	16
🔊 HTTP ( HyperText Transfer Protocol )	16
🔊 HTTP Request Message Format	17
🔊 Request Line	19
🔊 General Headers	19
🔊 Request Headers	19
Reference	22
Glossary	23

# Chapter 1 : Internet Protocol Suite



## Introduction to internet Protocol Suite

TCP/IP (Transmission Control Protocol/Internet Protocol) เป็นชุดของโปรโตคอลที่ใช้ในการสื่อสารผ่านเครือข่ายอินเทอร์เน็ต โดยมีวัตถุประสงค์เพื่อให้สามารถใช้สื่อสารจากต้นทางข้ามเครือข่ายไปยังปลายทางได้ และสามารถหาเส้นทางที่จะส่งข้อมูลไปตัวเองโดยอัตโนมัติ ถึงแม้ว่าในระหว่างทางอาจจะผ่านเครือข่ายที่มีปัญหา โปรโตคอลก็ยังค้นหาเส้นทางอื่นในการส่งผ่านข้อมูลไปให้ถึงปลายทางได้

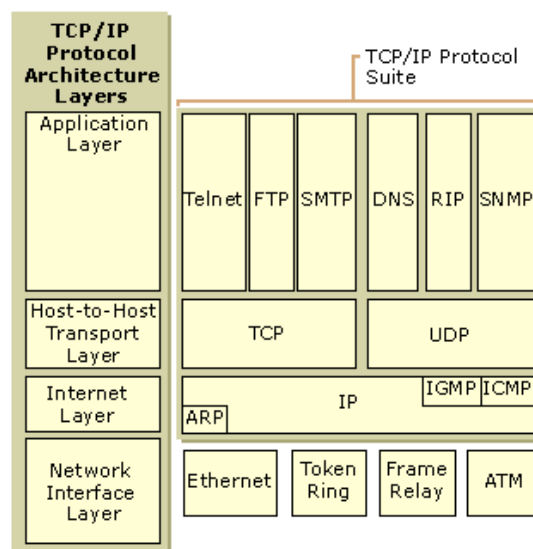
ชุดโปรโตคอลนี้ได้รับการพัฒนามาตั้งแต่ปี 1960 ซึ่งถูกใช้เป็นครั้งแรกในเครือข่าย ARPANET ซึ่งต่อมาได้ขยายการเชื่อมต่อไปทั่วโลกเป็นเครือข่ายอินเทอร์เน็ต ทำให้ TCP/IP เป็นที่ยอมรับอย่างกว้างขวางจนถึงปัจจุบัน



## TCP/IP Layer Description

ชุดอินเทอร์เน็ตโปรโตคอลมีการแบ่งการทำงานเป็นชั้นๆ (layer) เหมือนกับชุดโปรโตคอลอื่นๆ เพื่อแก้ปัญหาเฉพาะอย่าง เช่น ปัญหาเรื่องการส่งข้อมูล ปัญหาการจัดการข้อมูลเป็นต้น ซึ่งชุดอินเทอร์เน็ตโปรโตคอลแบ่งได้เป็น 4 ชั้น ได้แก่

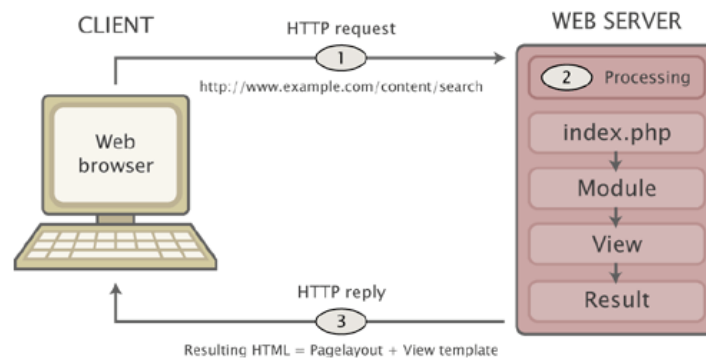
1. Application Layer
2. Transport Layer
3. Internet Layer
4. Network Interface Layer / Host-to-Network Layer



### 1. Application Layer

กำหนดรายละเอียดโปรโตคอลของโปรแกรม TCP/IP และวิธีการที่ host program จะติดต่อกับ Transport Layer เพื่อรับส่งข้อมูล ตัวอย่างโปรโตคอลในเลเยอร์นี้ได้แก่ HTTP FTP SMTP DNS เป็นต้น

## HTTP Example



### HTTP Request

```
GET / HTTP/1.1
Host: www.ku.ac.th
Connection: close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no
Accept-Language: de,en;q=0.7,en-us;q=0.3
```

### HTTP Response

```
HTTP/1.1 200 OK
Date: Fri, 01 May 2009 07:03:52 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=tis-620

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML><HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=tis-620">
<META NAME="Author" CONTENT="Kasetsart University,Thailand,Bangkok">
<META NAME="Keywords" lang="en" CONTENT="Kasetsart, kaset, thai , thailand , Academic, education ,
Kasetsart University,Thailand, University, thai University, Bangkok, Higher Education, Equality in Education,
Colleges, School, Campuses, Faculty, Technology, Engineering, Science, MBA, Research">
<META NAME="Keywords" lang="th" CONTENT="มหาวิทยาลัย , มหาวิทยาลัยเกษตรศาสตร์ , เกษตร,
..... Data.....
```

## 2. Transport Layer

จัดการเกี่ยว session การเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์และกำหนดขั้นตอนและสถานะการเชื่อมต่อเมื่อส่งข้อมูล โพรโทคอลที่ทำงานในชั้นนี้ได้แก่ TCP และ UDP

- TCP (Transmission Control Protocol)

เป็นโปรโตคอลสำหรับรับส่งข้อมูลซึ่งมีกลไกตรวจสอบความถูกต้องของข้อมูลที่ได้รับมาด้วย ทำให้ข้อมูลเชื่อถือได้ แต่มีข้อเสียคือ จะทำงานช้ากว่า UDP

- UDP (User Datagram Protocol)

เป็นโปรโตคอลสำหรับรับส่งข้อมูลเช่นเดียวกับ TCP แต่การส่งจะมีความยืดหยุ่นกว่าคือไม่จำเป็นต้องสร้างช่องทางการเชื่อมต่อไว้ แต่ข้อมูลอาจผิดพลาดได้เนื่องจากไม่มีกลไกตรวจสอบความถูกต้อง UDP นั้นนิยมใช้กับการส่งข้อมูลที่เป็นรีลไทม์แต่ไม่ต้องการความถูกต้องของข้อมูลมากนัก เช่น Game Online ต่างๆ

## 3. Internet Layer

จัดเก็บข้อมูลในรูปของ IP datagrams ซึ่งประกอบด้วยข้อมูลที่อยู่ของต้นทางและปลายทางเพื่อใช้ส่งข้อมูล (datagram) ระหว่างคอมพิวเตอร์ผ่านระบบเครือข่าย อีกทั้งหาเส้นทาง (routing) ของการส่งข้อมูลด้วยโปรโตคอลที่ทำงานในชั้นนี้ ได้แก่ IP, ICMP, ARP, RARP เป็นต้น

## 4. Network Interface Layer

ระบุรายละเอียดวิธีการที่ข้อมูลจะถูกส่งผ่านระบบเครือข่ายในระดับกายภาพ ตั้งแต่ข้อมูลระดับบิตแทนด้วยสัญญาณไฟฟ้าอย่างไร การเชื่อมต่อระหว่างตัวกลาง เช่น optic fiber , coaxial cable ทำอย่างไร



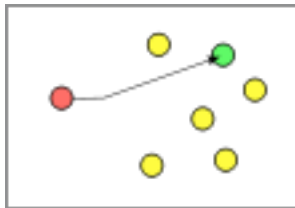
## IP (Internet Protocol) Address

ไอพีแอดเดรส เป็นตัวเลขที่ใช้ระบุถึงอุปกรณ์ต่างๆ ภายในระบบเครือข่าย ซึ่งปัจจุบันมี 2 แบบ คือ IPv4 ใช้ตัวเลขขนาด 32 บิตในการระบุ และ IPv6 ใช้ตัวเลขขนาด 128 บิตในการระบุ ตัวอย่าง 158.108.204.14 เป็น ไอพีของเครื่องภายในเครือข่ายของมหาวิทยาลัยเกษตรศาสตร์



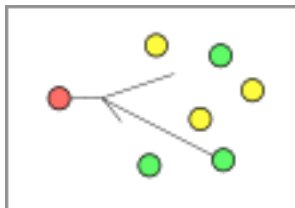
## Routing Schemes

Routing Scheme คือรูปแบบการส่งข้อมูลระหว่างโหนด (node) หรืออุปกรณ์ภายในระบบเครือข่าย



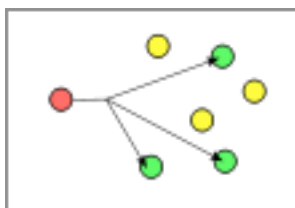
### Unicast

การส่งข้อมูลแบบ one-to-one



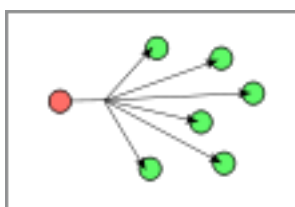
### Anycast

การส่งข้อมูลแบบ one-to-many แต่จะมีเครื่องเดียว ณ ขณะนั้นที่จะได้รับ โดยปกติจะเป็นเครื่องที่ใกล้ข้อมูลที่สุด



### Multicast

การส่งข้อมูลแบบ one-to-many โดยเครื่องที่จะรับข้อมูลต้องทำการลงทะเบียนไว้ก่อน



### Broadcast

การส่งข้อมูลแบบ one-to-many โดยจะส่งข้อมูลไปยังทุกเครื่องในเครือข่าย



# Chapter 2: Basic Socket Programming



## Introduction to Basic Socket Programming

Socket Programming คือการเขียนโปรแกรมโดยใช้ socket เพื่อติดต่อสื่อสารกันระหว่างโปรแกรมหรือโพรเซส (process) บนระบบเครือข่าย โดยโปรแกรมหรือโพรเซสที่เริ่มการติดต่อจะเรียกว่าไคลเอนต์ (client) ส่วนโปรแกรมหรือโพรเซสที่รอการติดต่อเข้ามาจะเรียกว่าเซิร์ฟเวอร์ (server) ซึ่งรูปแบบการติดต่อสื่อสารระหว่างไคลเอนต์และเซิร์ฟเวอร์จะเป็นแบบ connection-oriented หรือ connectionless ก็ได้

การเชื่อมต่อที่เครื่องแต่ละเครื่องสามารถเป็นได้ทั้ง client และ server จะเรียกว่าการเชื่อมต่อแบบ Peer-to-Peer (P2P) โปรโตคอลที่ทำงานแบบ Peer-to-Peer ได้แก่ Bittorrent และ Napster เป็นต้น ซึ่งเนื้อหาในบทนี้จะกล่าวเฉพาะการเขียนโปรแกรมกับ socket บน TCP และ UDP เท่านั้น



## Connection-oriented vs Connectionless

TCP จะเป็นการสื่อสารข้อมูลแบบ Connection-Oriented คือมีลักษณะเหมือนการส่งข้อมูลเสียงทางโทรศัพท์ คือผู้ใช้ต้องสร้าง connection (หมุนโทรศัพท์) แล้วถึงส่งข้อมูล (พูดโทรศัพท์) และเมื่อใช้เสร็จแล้วก็ยกเลิก connection (วางสายโทรศัพท์) การส่งข้อมูลแบบนี้ เปรียบเสมือนส่งของผ่านท่อ คือผู้ส่งส่งของทีละชิ้นไปตามท่อ แล้วผู้รับซึ่งอยู่อีกปลายหนึ่งของท่อก็รับของทีละชิ้นออกจากท่อ ตามลำดับที่ของถูกส่งมา

TCP ซึ่งเป็นแบบ Connection-Oriented นี้ จะต้องเสียเวลาในการเริ่มต้นทำการสื่อสารค่อนข้างนาน การรับส่งข้อมูลจะมีความถูกต้อง และรับรองการได้รับของอีกฝ่ายได้แน่นอน โดยผู้ส่งจะรอรับคำยืนยันว่า "ได้รับแล้ว" ของข้อมูลชุดที่แล้วจากผู้รับเสียก่อน จึงค่อยดำเนินการส่งข้อมูลชุดต่อไป เหมาะกับข้อมูลปริมาณมากๆ และมีความสำคัญ ตัวอย่างการใช้งานที่ใช้ TCP เช่น E-mail , World Wide Web และ FTP (File Transfer Protocol) เป็นต้น

สำหรับแบบ UDP จะเป็นการสื่อสารข้อมูลอีกชนิดหนึ่งที่เรารู้จักว่า Connectionless แบบนี้มีลักษณะคล้ายการส่งจดหมาย ในระบบไปรษณีย์ กล่าวคือข้อมูลหน่วยย่อย (จดหมายแต่ละฉบับ) มีที่อยู่ปลายทางของผู้รับ และแต่ละหน่วยข้อมูลจะถูกส่งต่อเป็นช่วงๆ (ผ่านที่ทำการไปรษณีย์แต่ละพื้นที่) จนถึงจุดหมาย การส่งข้อมูลลักษณะนี้แต่ละหน่วยข้อมูลอาจมีเส้นทางต่างกันเล็กน้อย และเป็นไปได้ว่าจดหมายที่ส่งทีหลังอาจถึงปลายทางก่อน

แบบ Connectionless นี้ การเริ่มต้นส่งสามารถทำได้รวดเร็ว เนื่องจากไม่ต้องเสียเวลา สร้าง connection แต่ก็ไม่สามารถรับรองการได้รับข้อมูลของอีกฝ่าย เหมาะกับการส่งข้อมูลเพียงเล็กน้อย ส่งเพียงครั้งเดียวก็เสร็จสิ้น หรือข้อมูลที่ไม่สำคัญมาก สามารถสูญเสียได้บางส่วน ตัวอย่างงานที่ใช้ UDP เช่น สัญญาณ Video , เสียง ซึ่งข้อมูลสามารถหายไปบางส่วนได้

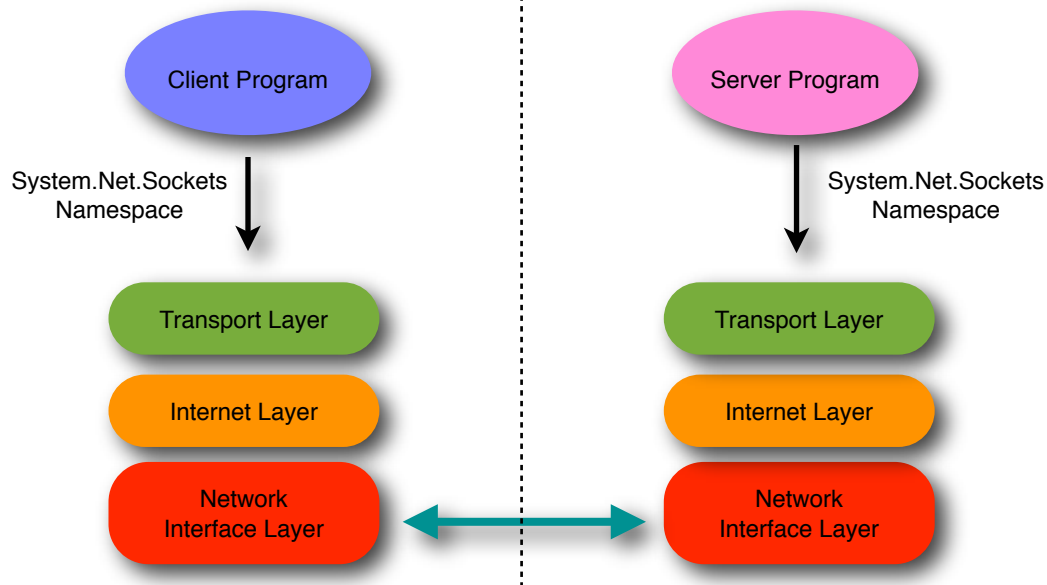


## Synchronous vs Asynchronous



## System.Net.Sockets Namespace

System.Net.Sockets Namespace ได้รวบรวมคลาสต่างๆ ที่เกี่ยวกับการเขียนโปรแกรมกับ socket ไว้ เพื่อให้ นักพัฒนาได้ใช้เมื่อต้องการเชื่อมต่อระบบเครือข่ายอย่างมีประสิทธิภาพ เพราะผู้พัฒนาสามารถออกแบบ โปรโตคอลประยุกต์ (Application Protocol) ได้เอง



ภาพแสดงถึงความสัมพันธ์ของ System.Net.Sockets Namespace กับชั้นการทำงาน TCP/IP

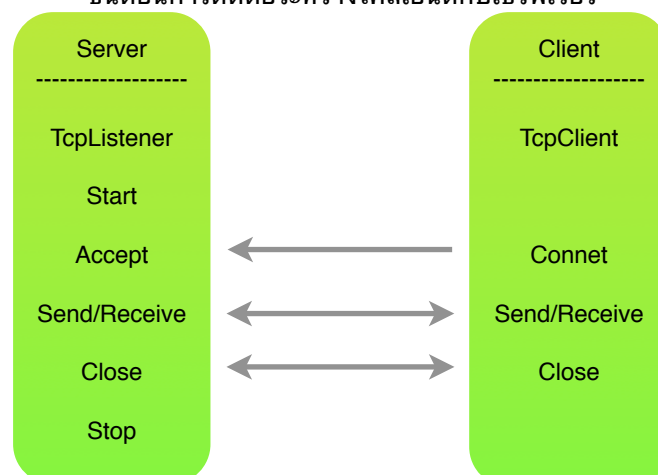
คลาสที่น่าสนใจในเนมสเปซนี้

Class	Description
NetworkStream	The NetworkStream class allows network data to be read and written in the same manner as the Stream class.
TcpClient	Provides client connections for TCP network services
TcpListener	Listens for connections from TCP network clients
UdpClient	Provides User Datagram Protocol (UDP) network services



## How to make a connection ( Synchronous )

ขั้นตอนการติดต่อระหว่างไคลเอนต์กับเซิร์ฟเวอร์





## ตัวอย่างโปรแกรม MakeConnection Server

```
using System;  
using System.IO;  
using System.Net;  
using System.Net.Sockets;
```

Namespace ที่ต้องใช้

```
class Server  
{  
    static void Main(string[] args)  
    {  
        int port = 6112;  
        IPAddress localAddr = IPAddress.Parse("127.0.0.1");  
        TcpListener tcpListener = new TcpListener(localAddr, port);
```

### Step 1

สร้าง TcpListener object  
พร้อมกำหนด local IP และ port ที่จะ  
รอการติดต่อจาก client

```
try  
{
```

```
    tcpListener.Start();  
    Console.WriteLine("Waiting for client");
```

Step 2 รอการเชื่อมต่อจาก client

```
    TcpClient tcpClient = tcpListener.AcceptTcpClient();  
    Console.WriteLine("Accept incoming connection");
```

Step 3 ยอมรับการเชื่อมต่อจาก client

```
    // แสดง IP address ที่ติดต่อเข้ามา  
    Console.WriteLine("The connection is from {0}", ((IPEndPoint)tcpClient.Client.LocalEndPoint).Address);
```

```
    tcpClient.Close();  
    Console.WriteLine("Close connection");
```

Step 4 ตัดการเชื่อมต่อ

```
    tcpListener.Stop();  
    Console.WriteLine("Stop waiting for client");  
}  
catch (Exception e)  
{  
    Console.WriteLine(e.ToString()); // แสดงข้อผิดพลาดที่เกิดขึ้น  
}  
Console.WriteLine("Press any keys to terminate program");  
Console.Read();  
}
```

Step 5 ยกเลิกการรอการเชื่อมต่อจาก client

## ผลลัพธ์ MakeConnection Server

```
Waiting for client  
Accept incoming connection  
The connection is from 127.0.0.1  
Close connection  
Stop waiting for client  
Press any key to terminate program
```

## ตัวอย่างโปรแกรม MakeConnection Client

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
```

Namespace ที่ต้องใช้

```
class Client
{
    static void Main(string[] args)
    {
        int port = 6112;
        IPAddress serverAddr = IPAddress.Parse("127.0.0.1");
        TcpClient tcpClient = new TcpClient();

        try
        {
            tcpClient.Connect(serverAddr, port);
            Console.WriteLine("Connect to server from given IP/Port ");

            if (tcpClient.Connected == true)
            {
                Console.WriteLine("Server accepted the connection");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.ToString());
        }
        finally
        {
            tcpClient.Close();
        }

        Console.WriteLine("Press any keys to terminate program");
        Console.Read();
    }
}
```

### Step 1

สร้าง TcpClient object

Step 2 ติดต่อไปยัง server จาก ip และ port ที่กำหนด

Step 3 ตัดการเชื่อมต่อ

## ผลลัพธ์ MakeConnection Client

```
Connect to server from given IP/Port
Server accepted the connection
Press any keys to terminate program
```



## How to send and receive data

### วิธีที่ 1 การรับส่งข้อมูลโดยใช้ **NetworkStream**

หลังจากทำการเชื่อมต่อสำเร็จแล้ว ฝั่งเซิร์ฟเวอร์และไคลเอนต์จะทำการสร้าง **NetworkStream** ซึ่งใช้สำหรับรับและส่งข้อมูลระหว่างกัน

#### ขั้นตอนการส่ง

1. รับ **NetworkStream** โดยเรียก **GetStream()** method
2. แปลงข้อมูลที่ต้องการเป็น **Byte array**
3. ส่ง **Byte array** ที่แปลงผ่านทาง **Network Stream**

#### ขั้นตอนการรับ

1. รับ **NetworkStream** โดยเรียก **GetStream()** method
2. สร้างบัฟเฟอร์เพื่อรับข้อมูล
3. รับข้อมูลจาก **NetworkStream** เป็น **Byte array**
4. แปลง **Byte array** เป็นชนิดข้อมูลที่ต้องการ

**NetworkStream** จะมีเมธอด **Write** เพื่อเขียนข้อมูลลงบัฟเฟอร์ เมธอด **Read** เพื่ออ่านข้อมูลจากบัฟเฟอร์ และ **Flush** เพื่อส่งข้อมูลจากบัฟเฟอร์ไปยังปลายทาง  
การใช้งานเมธอด **Write**

```
void Write ( Byte[] buffer , int offset , int size ) ;
```

**buffer** คือ ข้อมูลที่จะเขียนซึ่งเป็นอาร์เรย์ของไบต์  
**offset** คือ ตำแหน่งข้อมูลในอาร์เรย์ที่จะเริ่มเขียน  
**size** คือ จำนวนข้อมูลที่ต้องการเขียน

#### การใช้งานเมธอด **Read**

```
int Read ( Byte[] buffer , int offset , int size ) ;
```

**buffer** คือ ที่เก็บข้อมูลที่จะอ่านซึ่งเป็นอาร์เรย์ของไบต์  
**offset** คือ ตำแหน่งเริ่มต้นในอาร์เรย์ที่จะเก็บข้อมูล  
**size** คือ จำนวนข้อมูลที่ต้องการอ่าน  
โดยจะคืนค่าจำไบต์ที่อ่านได้

การรับส่งข้อมูลพื้นฐาน (**boolean, short, char, int, long, float, double**) จะใช้เมธอด **GetBytes()** ของคลาส **BitConverter** ในการแปลงข้อมูลจากข้อมูลชนิดพื้นฐานเป็นอาร์เรย์ของไบต์ และใช้เมธอด **ToDatatype** ในการแปลงอาร์เรย์ของไบต์เป็นข้อมูลชนิดพื้นฐานนั้นๆ

#### การใช้เมธอด **GetBytes**

```
byte[] GetBytes ( data_type data ) ;
```

**data\_type** ที่สามารถใช้ได้คือ **boolean, short, char, int, uint, ulong, long, float, double, string**  
**data** คือ ข้อมูลที่ต้องการแปลง

## ตัวอย่างโปรแกรม SendReceive Client

```
short myShort = 128;
bool myBool = true;
char myChar = 'A';
int myInt = 1024;
uint myUInt = 2048;
long myLong = 65536;
float myFloat = 3.14159f;
double myDouble = 1.618;
```

ข้อมูลชนิดพื้นฐานต่างๆ

```
NetworkStream networkStream = tcpClient.GetStream();
```

step 1 รับ NetworkStream

```
byte[] myShortByte = BitConverter.GetBytes(myShort);
byte[] myBoolByte = BitConverter.GetBytes(myBool);
byte[] myCharByte = BitConverter.GetBytes(myChar);
byte[] myIntByte = BitConverter.GetBytes(myInt);
byte[] myUIntByte = BitConverter.GetBytes(myUInt);
byte[] myLongByte = BitConverter.GetBytes(myLong);
byte[] myFloatByte = BitConverter.GetBytes(myFloat);
byte[] myDoubleByte = BitConverter.GetBytes(myDouble);
```

step 2 แปลงข้อมูลพื้นฐานเป็นอาเรย์ของไบนารี

```
networkStream.Write(myShortByte, 0, myShortByte.Length);
networkStream.Write(myBoolByte, 0, myBoolByte.Length);
networkStream.Write(myCharByte, 0, myCharByte.Length);
networkStream.Write(myIntByte, 0, myIntByte.Length);
networkStream.Write(myUIntByte, 0, myUIntByte.Length);
networkStream.Write(myLongByte, 0, myLongByte.Length);
networkStream.Write(myFloatByte, 0, myFloatByte.Length);
networkStream.Write(myDoubleByte, 0, myDoubleByte.Length);
```

step 3 ส่งข้อมูลผ่านโดยใช้ NetworkStream

## ตัวอย่างโปรแกรม SendReceive Server

```
NetworkStream networkStream = tcpClient.GetStream();
int count;
```

step 1 รับ NetworkStream

```
byte[] buffer = new byte[16];
```

step 2 สร้างบัฟเฟอร์เพื่อเก็บข้อมูล  
step 3 อ่านข้อมูลจาก NetworkStream

```
count = networkStream.Read(buffer, 0, sizeof(short));
Console.WriteLine("Short \tfrom client : {0}\tByte recived : {1}", BitConverter.ToInt16(buffer, 0), count);
```

step 4  
แปลง  
ข้อมูลเป็น  
ชนิดที่  
ต้องการ

```
count = networkStream.Read(buffer, 0, sizeof(bool));
Console.WriteLine("Bool \tfrom client : {0}\tByte recived : {1}", BitConverter.ToBoolean(buffer, 0), count);
```

```
count = networkStream.Read(buffer, 0, sizeof(char));
Console.WriteLine("Char \tfrom client : {0}\tByte recived : {1}", BitConverter.ToChar(buffer, 0), count);
```

```
count = networkStream.Read(buffer, 0, sizeof(int));
Console.WriteLine("Int \tfrom client : {0}\tByte recived : {1}", BitConverter.ToInt32(buffer, 0), count);
```

```
count = networkStream.Read(buffer, 0, sizeof(uint));
Console.WriteLine("UInt \tfrom client : {0}\tByte recived : {1}", BitConverter.ToUInt32(buffer, 0), count);
```

.....

จำนวน  
ไบนารีที่  
อ่านได้

การส่งสตริงผ่านเน็ตเวิร์กเราจะแปลงข้อความเป็นไบนารีโดยใช้คลาส Encoding ในการแปลง ซึ่งทำให้เราสามารถเข้ารหัสให้เหมาะสมกับภาษาได้ รหัสที่นิยมใช้ได้แก่

ASCII สำหรับข้อความที่ประกอบด้วยอักษรภาษาอังกฤษเท่านั้น  
Unicode สำหรับข้อความที่ประกอบด้วยอักษรหลายภาษา

การแปลงข้อความเป็นไบนารี

```
byte[] Encoding.ASCII.GetBytes(String data); // ASCII encoding
byte[] Encoding.Unicode.GetBytes(String data); // Unicode encoding
```

เมื่อ data คือ สตริงที่ต้องการจะแปลงเป็นไบนารี

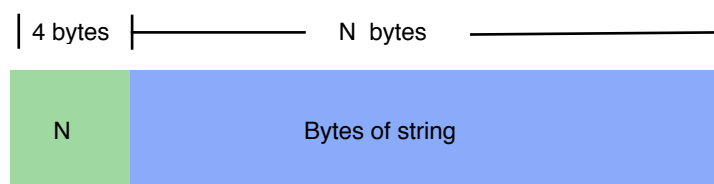
การแปลงไบนารีเป็นข้อความ

```
string Encoding.ASCII.GetString(byte[] buffer, int offset, int size); // ASCII encoding
string Encoding.Unicode.GetString(byte[] buffer, int offset, int size); // Unicode encoding
```

buffer คือ ข้อมูลที่จะแปลงเป็นข้อความ  
offset คือ ตำแหน่งข้อมูลในอาร์เรย์ที่จะเริ่มแปลง  
size คือ จำนวนข้อมูลที่ต้องการแปลง

ขั้นตอนการรับส่งสตริงโดยใช้ NetworkStream

เนื่องจากสตริงจะต่างจากข้อมูลชนิดพื้นฐานอื่นๆ ตรงที่ขนาดข้อมูลไม่คงที่ ดังนั้นการส่งข้อมูลไปยังปลายทางโดยใช้ NetworkStream เราจะทำการส่งขนาดของไบนารีของสตริงไปก่อน แล้วตามด้วยไบนารีของสตริง เพื่อให้ปลายทางสามารถอ่านข้อมูลสตริงลงบัฟเฟอร์ได้ถูกต้อง



ลักษณะข้อมูลสตริงที่ส่งในระดับไบนารี

ตัวอย่างโปรแกรม SendReceive Client

```
string myString = "eXceed Camp #6";
byte[] myStringByte = Encoding.Unicode.GetBytes(myString);
byte[] byteRead = BitConverter.GetBytes( myStringByte.Length);
```

```
networkStream.Write(byteRead, 0, byteRead.Length);
networkStream.Write(myStringByte, 0, myStringByte.Length);
```

แปลงข้อความเป็นไบนารีโดยเข้ารหัสแบบ unicode  
หาไบนารีของขนาดไบนารีของข้อความที่แปลง

ส่งขนาดไบนารี  
ส่งไบนารีของข้อความที่เข้ารหัส

### ตัวอย่างโปรแกรม MakeConnection Server

จำนวน  
ไบต์อาเรย์  
ของ  
ข้อความ

```
networkStream.Read(buffer, 0, sizeof(int));  
int readByte = BitConverter.ToInt32(buffer,0);
```

อ่านขนาดไบต์อาเรย์ของข้อความ

อ่านไบต์อาเรย์ของข้อความตามจำนวนไบต์ที่ส่งมา  
แปลงไบต์อาเรย์เป็นข้อความตามจำนวนไบต์ที่ส่งมา

```
networkStream.Read(buffer, 0, readByte);  
string myString = Encoding.Unicode.GetString(buffer,0,readByte);
```

## วิธีที่ 2 การรับส่งข้อมูลโดยใช้ StreamReader/StreamWriter

หากโปรแกรมที่เราออกแบบไว้ นั้น ติดต่อสื่อสารกันโดยใช้เฉพาะข้อความเท่านั้น เราสามารถใช้ StreamReader/StreamWriter แทนได้ ซึ่งจะทำให้การรับส่งข้อมูลทำได้ง่ายกว่าการใช้ NetworkStream โดยตรง

StreamReader เป็นคลาสสำหรับอ่านสตริง มีเมธอดที่น่าสนใจคือ

```
string ReadLine(); // ทำงานเช่นเดียวกับ ReadLine ของคลาส Console
```

StreamReader เป็นคลาสสำหรับเขียนสตริง มีเมธอดที่น่าสนใจคือ

```
string WriteLine( String str, object ob1 ...); // ทำงานเช่นเดียวกับ WriteLine ของคลาส  
// Console
```

```
string Flush(); // ส่งข้อมูลที่อยู่ในบัฟเฟอร์ไปยังปลายทางทันที
```

### ตัวอย่างโปรแกรม StreamReader StreamWriter Client

```
NetworkStream networkStream = tcpClient.GetStream();  
StreamWriter streamWriter = new StreamWriter(networkStream);
```

สร้าง StreamWriter

```
string command = Console.ReadLine();  
while (!command.ToLower().Equals("bye"))  
{  
    streamWriter.WriteLine(command);  
    streamWriter.Flush();  
    command = Console.ReadLine();  
}
```

เขียนข้อมูลลงบัฟเฟอร์โดยใช้ WriteLine  
ส่งข้อมูลที่อยู่ในบัฟเฟอร์ไปยังปลายทางทันที โดยใช้ Flush

```
streamWriter.WriteLine("bye");  
streamWriter.Flush();
```

```
networkStream.Close();
```

ปิดสตรีม

## ตัวอย่างโปรแกรม CH2 StreamReader StreamWriter Server

```
NetworkStream networkStream = tcpClient.GetStream();
StreamReader streamReader = new StreamReader(networkStream);
string myString;

do
{
    myString = streamReader.ReadLine();
    Console.WriteLine(myString);
}
while(!myString.ToLower().Equals("bye"));
...
networkStream.Close();
```

สร้าง StreamReader โดยรับ networkStream  
เป็น parameter

อ่านข้อมูลโดยใช้เมธอด ReadLine

ปิด NetWorkStream

# Chapter 3: Application Protocol



## Introduction to Application Protocol

หลังจากที่ได้ศึกษาการเขียนโปรแกรมเชื่อมต่อระหว่างคอมพิวเตอร์โดยใช้ TcpListener และ TcpClient ในบทที่ 2 ซึ่งเป็นการทำงานบน Transport Layer ที่ดูแลการเชื่อมและการรับส่งข้อมูลให้ถูกต้อง โดยการรับส่งข้อมูลนั้นจะอยู่ในรูปของไบต์สตรีม

ในบทนี้เราจะศึกษาโปรโตคอลหรือวิธีการติดต่อในระดับที่สูงกว่าไบต์ โปรโตคอลจะเป็นตัวกำหนดทั้งเซิร์ฟเวอร์และไคลเอนต์ว่าต้องส่งข้อมูลแบบไหน เวลาไหน ลำดับหน้าหลังเป็นอย่างไร เมื่อทั้งเซิร์ฟเวอร์และไคลเอนต์ปฏิบัติตามข้อกำหนดซึ่งก็คือโปรโตคอล ก็ทำให้สื่อสารกันได้สำเร็จ

Application protocol ที่น่าสนใจได้แก่

- HTTP ( 80 ) สำหรับรับส่งข้อมูลเกี่ยวกับเว็บไซต์
- FTP ( 21 ) สำหรับรับส่งไฟล์ระหว่างคอมพิวเตอร์
- SMTP ( 25 , 465 , 587 ) สำหรับส่งเมลล์
- POP3 ( 110 ) สำหรับรับเมลล์

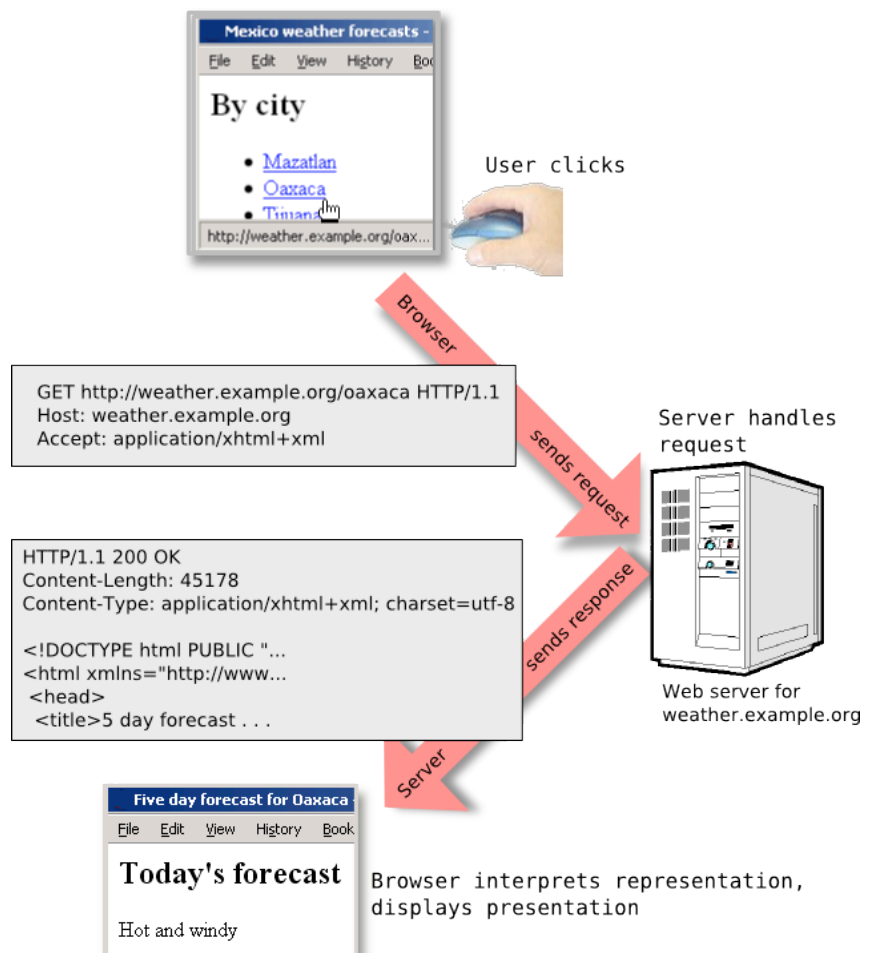


## HTTP ( HyperText Transfer Protocol )

HTTP เป็นโปรโตคอลที่ทำงานในชั้นแอปพลิเคชันของ TCP/IP model ซึ่งข้อมูลที่สื่อสารนั้นเป็นลักษณะเคลียร์เท็ก (cleartext) กล่าวคือเป็นข้อความที่มนุษย์สามารถอ่านแล้วเข้าใจได้ทันที

ขั้นตอนในการสื่อสารคือ ตัวเซิร์ฟเวอร์โดยทั่วไปเรียกเว็บเซิร์ฟเวอร์ จะรอการที่ต่อจากไคลเอนต์ที่พอร์ตมาตรฐาน คือ 80 เมื่อไคลเอนต์ติดต่อเข้ามาแล้ว จะทำการส่ง HTTP Request Message ไปยังเซิร์ฟเวอร์ จากนั้นเซิร์ฟเวอร์รับ HTTP Request Message ไปประมวล ว่าไคลเอนต์ต้องการเปิดหน้าไหน ส่งข้อมูลในฟอร์มอะไรมาบ้าง เมื่อประมวลผลเสร็จแล้วก็จะส่ง

HTTP Response Message ไปให้ไคลเอนต์ทำการแสดงผล

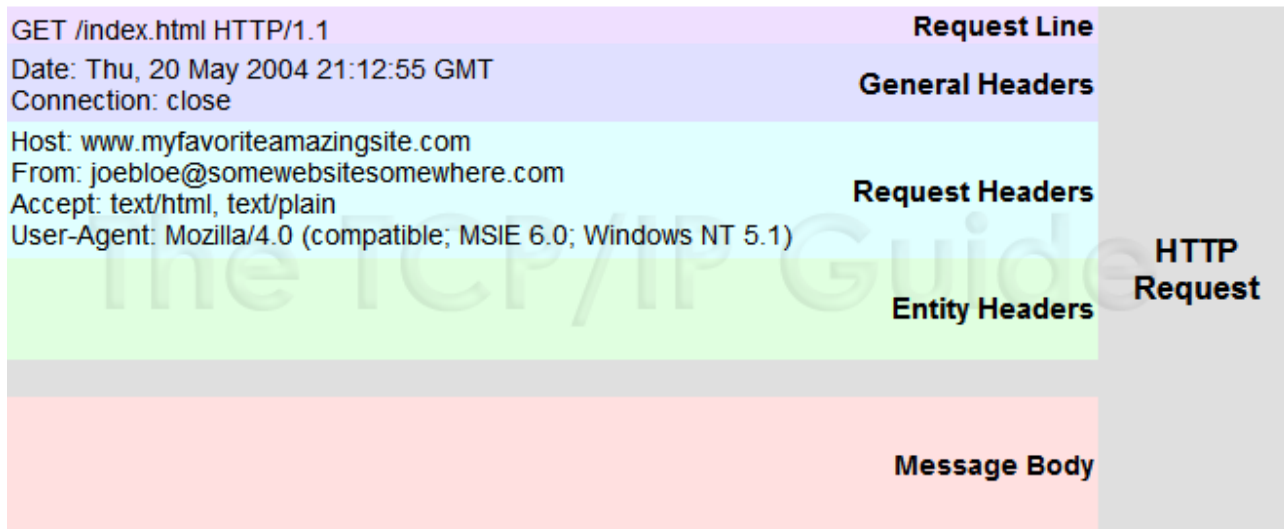






## HTTP Request Message Format

HTTP message ประกอบด้วย 2 ส่วนคือ ส่วนหัว (header) และส่วนข้อมูล (message body) ซึ่งทั้งสองจะคั่นด้วยสัญลักษณ์ <CR><LF> โดย CR ('\r') Carriage Return และ LF ('\n') Line Feed



ตัวอย่าง HTTP Request Message

```
GET / HTTP/1.0\r\n\r\n
```

เป็นข้อความสำหรับขอข้อมูลโฮมเพจของเว็บไซต์

ตัวอย่าง HTTP Response หลังจากส่ง Request Message "GET / HTTP/1.1\r\n" ไปยังโฮสต์ ku.ac.th ที่พอร์ต 80

```
HTTP/1.1 200 OK
Date: Fri, 01 May 2009 07:03:52 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=tis-620
```

บรรทัดตรงนี้เป็น <CR><LF>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML><HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=tis-620">
<META NAME="Author" CONTENT="Kasetsart University,Thailand,Bangkok">
<META NAME="Keywords" lang="en" CONTENT="Kasetsart, kaset, thai , thailand , Academic, education , Kasetsart
..... Data.....
```

โปรแกรม GetHomepage จะรับที่อยู่ของเว็บไซต์จากผู้ใช้ แล้วนำไปดึงข้อมูลหน้า Homepage มาแสดงผลออกทางหน้าจอ Console โดยการแปลงสตริง url เป็นออบเจ็กต์ IPAddress นั้นเราจะใช้เมธอด GetHostAddress() ของคลาส Dns

```
IPAddress[] Dns.GetHostAddress ( String url )
```

url คือ สตริงที่ต้องการจะแปลง ซึ่งสามารถอยู่ในรูป host name ( "www.google.co.th" )  
หรือ host ip ( "158.108.216.5" )

### ตัวอย่างโปรแกรม Get Homepage

```
using System;  
using System.IO;  
using System.Net;  
using System.Net.Sockets;
```

Required namespace

```
class GetHomePage  
{
```

```
    static void Main(string[] args)  
    {
```

```
        IPAddress[] ipAddress;  
        string url = "";  
        string request = "GET / HTTP/1.0\r\n\r\n";  
        string response;
```

HTTP Request Message

```
        Console.WriteLine("Enter Address( enter 'Bye' to exit ) : ");  
        while(!(url = Console.ReadLine()).Equals("bye",StringComparison.OrdinalIgnoreCase))  
        {
```

ทำการอ่าน url จากผู้ใช้  
จนกว่าจะพิมพ์ คำว่า 'bye'

ทำการแปลงชื่อเว็บไซต์  
หรือไอพี เป็นออบเจ็กต์ IPAddress

```
            try  
            {
```

```
                ipAddress = Dns.GetHostAddresses(url);  
                TcpClient client = new TcpClient();  
                client.Connect(ipAddress, 80);  
                NetworkStream networkStream = client.GetStream();  
                StreamWriter streamWriter = new StreamWriter(networkStream);  
                StreamReader streamReader = new StreamReader(networkStream);
```

เชื่อมต่อไปยังเป้าหมาย

```
                streamWriter.Write(request);  
                streamWriter.Flush();
```

ส่ง HTTP Request Message

```
                response = streamReader.ReadToEnd();  
                Console.WriteLine(response);
```

รับ HTTP Response Message แล้วแสดงผล

```
                networkStream.Close();  
                client.Close();
```

ปิดสตรีม

```
            }  
            catch(Exception e)
```

```
            {  
                Console.WriteLine(e);
```

```
            }  
            Console.WriteLine("Enter Address( enter 'Bye' to exit ) : ");
```

```
        }  
    }
```

```
}
```



## Request Line

Request Line จะเป็นส่วนที่ร้องขอข้อมูลหน้าเว็บเพจจากเว็บเซิร์ฟเวอร์ โดยมีรูปแบบการใช้คือ

Method Url HTTP/1.0

Method คือ วิธีการร้องขอข้อมูล ซึ่งประกอบด้วย GET และ POST

Url คือ ตำแหน่งที่ของไฟล์บนเซิร์ฟเวอร์ โดย '/' หมายถึงโฮมเพจ และ Url ยังสามารถแทรก query string ได้ด้วย เช่น "/index.html?param1=1&param2=2"

GET /Test/hello.html HTTP/1.0



## General Headers

General headers เป็นส่วนเป็นส่วนของเฮดเดอร์ที่มีทั้งใน Request และ Response message ฟیلด์ (field) ที่แนะนำคือ Date

Date Field เป็นส่วนที่บอกว่า HTTP Message นั้นถูกส่งไปเมื่อใด

Date: Tue, 15 Nov 1994 08:12:31 GMT



## Request Headers

Request headers เป็นฟیلด์ที่จะส่งข้อมูลเพิ่มเติมเกี่ยวกับไคลเอนต์ไปยังเซิร์ฟเวอร์

From Field เป็นอีเมลล์ของผู้ใช้ที่ทำการส่ง Request Message

From: webmaster@w3.org

Referer Field เป็นค่าที่บอกเซิร์ฟเวอร์ว่าก่อนที่จะร้องขอหน้านี้ เคยเข้าหน้าไหนมาก่อนแล้ว ซึ่งค่าเป็นได้ทั้ง absolute path หรือ relative path

Referer: http://www.w3.org/hypertext/DataSources/Overview.html

User-Agent เป็นชื่อของบราวเซอร์และระบบปฏิบัติการของผู้ใช้

User-Agent: Mozilla/5.001 (windows; U; NT4.0; en-US; rv:1.0) Gecko/25250101

Mozilla Firefox บน Windows XP

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10\_5\_6; en-us) AppleWebKit/528.16 (KHTML, like Gecko) Version/4.0 Safari/528.16

Safari บน Mac OSX Leopard

## ตัวอย่าง HTTP Request Message

```
GET / HTTP/1.0
Date: Tue, 15 Nov 1994 08:12:31 GMT
From: swagen.xivth@gmail.com
Referer: http://www.ku.ac.th
User-Agent: Mozilla/5.001 (windows; U; NT4.0; en-US; rv:1.0) Gecko/25250101
```



# Reference

## Introduction to Internet Protocol Suite

- <http://www.thaicert.org/paper/basic/tcp-ip.php>
- <http://technet.microsoft.com/en-us/library/cc786900.aspx>

## Basic Socket Programming

- <http://www.sa.ac.th/e-learning/internet/connection.html>

## Application Protocol

- [http://www.tcpguide.com/free/t\\_HTTPRequestMessageFormat.htm](http://www.tcpguide.com/free/t_HTTPRequestMessageFormat.htm)
- <http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf>

# Glossary

protocol  
host program  
session  
Socket  
process  
stream